Langara College
CPSC 2150
Assignment #6: Binary Search Trees

Assignment due with Brightspace at 11:50pm on March 21

Drozdek §6.1, §6.2, §6.3, §6.4, §6.5, §6.6
Goodrich §7.3, §10.1

Purpose

- The purpose of this assignment is to implement a C++ class called Polynomial to represent a univariate polynomial with integer coefficients.
- To manipulate a Binary Search Tree: insert, delete, traverse a tree of terms.
- To integrate code into an existing program, understanding other people's code.
- To use an interface for a program: not just line arguments but possible redirection of a file but using a menu style interface that can be run "in batch".
- To use functors and lambda functions in C++17.

Background

A univariate polynomial is a polynomial in one variable. We will use x as the variable in this description. A univariate polynomial has the form

$$a_1\, x^{d1} + a_2\, x^{d2} + \ldots + a_n\, x^{dn}$$

where $n \geq 0$,
$a_1, a_2, \ldots a_n$ are integers and $d_1, d_2, \ldots d_n$ are distinct non-negative integers.
For example, the polynomial

$$8\, x^4 \;+\; 7\, x^2 \;-\, 3x^5 \;+\; 6$$

has $n = 4$ terms. The degree of this polynomial is 5 which is the highest power. Also, the coefficient of the term of degree 4 is 8, the coefficient of x^3 is 0.

Functionality

Implement an Abstract Data Type Polynomial that provides the following functionality (as per Polynomial.h that has many comments/instructions)

- creates a polynomial, the zero polynomial
- creates a polynomial with one term given a coefficient and a degree
- takes a term and adds it to the polynomial
  inserts the term into the correct place in the polynomial
- destroys a polynomial
- determines if the polynomial is the zero polynomial
- prints the polynomial using an output stream
- reads the polynomial from an input stream and builds a proper polynomial
- evaluates the polynomial for a given value of x
- gives the degree of the polynomial (the degree of a polynomial is the exponent of the term with the highest degree)
  as a special case, give -1 as the degree of the zero polynomial
- returns the coefficient of a term in the polynomial given a degree (given an exponent)
- returns the number of terms in the polynomial
- returns a new polynomial which is the sum of two polynomials
- changes the variable for outputting from x to another variable

Implementation

- use a **binary search tree** to store the terms by degree using the compare function passed to determine whether to insert into the left or right subtree (as documented in Polynomial.h)
- a tree node must have a pointer to a term (which stores the coefficient of the term and the degree of the term) and a pointer to each 'child node'.
- overload the + operator for addition: the sum is a new polynomial
- overload << for printing and >> for reading
- provide a copy constructor, destructor and overload the assignment operator
- free up heap memory that is no longer needed (there should be no memory leaks so deallocate terms no longer needed too)
- you should not have any terms (nor nodes for that matter) that have zero as the term coefficient (though you need to decide on a representation for the zero polynomial which is the constant 0)
- you should store only once in the tree a term of degree k:
  no two terms in the tree both of degree k
- you should be able to delete any node from the tree when needed:
  you will need this functionality when adding polynomials

- pass the boolean function that compares two degrees (to know how to insert into the binary search tree) as an argument to the constructors
- when printing the polynomial, print the binary search tree the way it was built printing from left to right
- use the Term class provided including the overloaded operator << for output (found in the files Term.h and Term.cpp)
  use the accessor and mutator functions of Term as needed
- use the Polynomial.h and Polynomial.cpp provided for you (including the given definition of Node though you may change to a class) and expand it i.e. add code as needed in both Polynomial.h and Polynomial.cpp and document your code. So, to reiterate, **add** code to both Polynomial.h and Polynomial.cpp but do not modify the interface.

Example

In the input data
5
8 4
7 2
-3 5
6 0
-1 4

5 refers to the number of terms and the terms corresponding to the values given above are
$8\ x^4$
$7\ x^2$
$-\ 3x^5$
6
$-x^4$
Note that after the terms are read into an instance of the Polynomial class which internally is represented with a Binary Search Tree, the polynomial will have only 4 tree nodes (not 5) i.e. the number of terms is 4.
If on input the term $8\ x^4$ is given and then subsequently the term $-x^4$ because both terms have the same degree, in the BST there should only be one term namely $7\ x^4$.

The degree of the polynomial is 5
    - 3x^5 + 7x^4 + 7x^2 + 6

Calling the function 'coefficient' with the argument 4 returns 7 because the coefficient of the term that has degree 4 (namely of $7\ x^4$) is 7

Calling the function 'evaluate with x=1' for the polynomial returns 17.

You need to enter the zero polynomial "somehow" so if on input all the coefficients of the polynomial are zero, then it's the zero polynomial.

Application File

The application file solver is a simple command driven (menu style) program that tests your Polynomial class.

We have provided a series of 'commands' and data in a file called cmds.txt but not as a starting point.
Input interactively when first testing your Polynomial using solver

To run solver with an input file e.g. cmd.txt and to output to output.txt, do

```
solver  -batch < cmds.txt > output.txt
```

the input redirection probably will not work in a Windows PowerShell

Optional / Bonus

Differentiate the polynomial with respect to x.
Return a new polynomial which is the derivative.

To submit as Assignment #6 as a single compressed zip file

i. an output file called **myoutput.txt** that gives the results from running solver with input from **cmds.txt** with the option '-batch'
   Note that the program solver must read from std::cin. Leave as is.
ii. the source code
   a. Polynomial.h add the missing code and other functions/variables.
      Please do not inline the code in the file Polynomial.h
      Do not put any function definitions in Polynomial.h.
      Yes, put function declarations (or function prototypes).
   b. Polynomial.cpp add the missing code and other functions/variables
   c. possibly Node.h and Node.cpp and an adjusted Makefile -- no need to make Node a class but you could
   d. the following files provided for you, please include them in your submission
   i. solver.cpp
   ii. Makefile to compile and link solver.cpp in C++17
   iii. Commands.cpp and Commands.h
   iv. Term.cpp and Term.h
      you may expand them but do not make the instance variables public: the output operator for a Term has been provided for you
   v. cmds.txt

iii. a file called README.txt if needed where
   a. you indicate that you implemented the BONUS part
   b. you can list there what is not implemented in the assignment in general (if you did not have time for certain parts)
   c. if you are not pleased with your design, you can write how you would design your program differently if you had more time

Notes

Implement yourself the insertion of the nodes and deletion and the copying and destroying of the BST and so on. Do not find a "library" that does it for you. Do not use the STL maps or sets (yes, they are used in solver.cpp).

Yes, you can use code provided in class and/or textbooks. **Cite** "class notes" or "Drozdek" or "Goodrich" e.g. the code for deleting a node. Otherwise, it's considered plagiarized code. Citing a friend or the Internet is not appropriate as it is not your code that you are submitting.

Please do not include in your code non-standardized C++17 e.g.
#pragma once
#include "pch.h"

Do include the header file of the library functions that you use e.g.
#include <cassert>
#include <cstdlib>
#include <cmath>

Do not use while(true) as a construct when programming.